ATTACHMENT 8

a) Delphi Alternator
Saber-based Program Listing

alternator (Alternator)

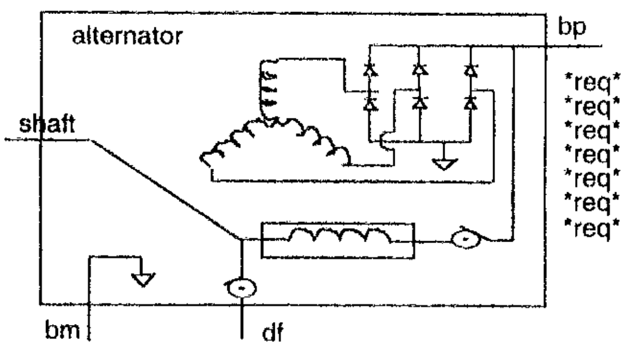
encrypted template alternator bp bm df shaft = rs, lls, lms, rr, llr, lmr, p, d, j, theta0, units, ratings, part_type, part_class

Associated Symbols: `alternator`

License Requirements: OPT_TEMPLATE_LIB or CUSTOM_LIB

Description

The **alternator** template models a complete electro-mechanical automotive alternator with the rectifying diodes. The model converts the mechanical energy provided by shaft to electrical output across bp and bm. The model includes all of the internal flux coupling factors that vary with the shaft rotation angle.



Alternator (alternator)

Connection Points	Symbol Properties	Stress Arguments
Post-Processing Information	External Variables	Usage Notes
Netlist Examples	References	

See Also

Part Category: Electro-mechanical Templates
Electrical Templates -- Automotive

alternator Connection Points

Name	Type	Description

```
#       3Phase Model of the AD-230 14V Generator
#       generator_delphi_ad230_3phase.sin

####
#       Model written by John MacBain based heavily on a model provided by
#       Mike Donnelly of Analogy Inc.
#       Present limitation is based upon one operating point.  Future version
#       will be nonlinear.  For the user's convenience, the nonlinear parameter
#       are detailed here:
#
#       shaft rpm       lms(uH)         kmult
#       1600            397             .563
#       1800            406             .569
#       2000            411             .573
#       2500            423             .580
#       3000            433             .587
#       3500            439             .592
#       4000            444             .595
#       5000            447             .597
#       6500            446             .596
#       8000            443             .594
#
#       This data was generated by Ron Krefta.  Temperature = 25C with the
#       generator stabilized at that temperature.
#
#       This generator is Delta wound.

template generator_delphi_ad230_3phase a b c fm fp shaft emf com = kmult, rs, l

electrical      a,      # Phase a output
                b,      # Phase b output
                c,      # Phase c output
                fm,     # Negative connection of the field winding
                fp,     # Positive connection of the field winding
                emf,
                com

var nu shaft # # rotor mechanical angular velocity connection
#rotational_vel shaft # rotor mechanical angular velocity connection
#               shaft should now be input as rpm - when wr is created from
#               shaft, wr is in radians per second required of trig functions

number          rs= 74m,        # Resistance of stator windings (Ohms).
                lls= 166u,      # Leakage inductance of stator windings (H).
                lms= 443u,      # Magnitizing inductance of stator windings (H
                rr=1.6799,      # Resistance of field winding (Ohms).
                llr=123.3m,     # Leakage inductance of field winding (H).
                lmr=291.8m,     # Magnitizing inductance of field winding (H).
                p = 12,         # Number of poles. (p must be even, >= 2)
                d=50u,          # Alternator viscous damping constant (N*m/(r/
                j=100u,         # Alternator shaft inertia.
                theta0=0,       # Initial rotor angle .
                kmult = .594    # Mutual Inductance Factor

#       Message("kmult = %",kmult)

mech_def..units units = mks     # Selects "units" assumed for arguments.
```

```
###########
###########    Internal 3 phase model, alt3p
###########
template alt3p asp bsp csp fp fm shaft emf com = kmult, rs,lls,lms,rr,llr,lmr,p,
                                               d,j,theta0,units # ,ifld_res

electrical       asp,     # phase a positive terminal
                 bsp,     # phase b positive terminal
                 csp,     # phase c positive terminal
                 fp,      # positive field (rotor) terminal
                 fm,      # negative field (rotor) terminal
                 emf,
                 com

var nu  shaft    # rotor mechanical angular velocity connection
#rotational_vel shaft    # rotor mechanical angular velocity connection

number           rs,      # Resistance of stator windings (Ohms).
                 lls,     # Leakage inductance of stator windings (H).
                 lms,     # Magnitizing inductance of stator windings (H).
                 rr,      # Resistance of field winding (Ohms).
                 llr,     # Leakage inductance of field winding (H).
                 lmr,     # Magnitizing inductance of field winding (H).
                 p,       # Number of poles. (p must be even, >= 2)
                 d,       # Alternator viscous damping constant (N*m/(r/s)).
                 j,       # Alternator shaft inertia.
                 theta0,  # Initial rotor angle .
                 kmult    # Mutual Inductance Factor

mech_def..units units = mks      # Selects "units" assumed for arguments.

#number ifld_res = 0      # Residual field flux (equiv current) for self-start.

{
struc{
      number bp,inc
      } sp[*]             # Sample point array for rotor electrical angle.
struc{
      number bp,inc
      } ns[*]             # Newton Step array for rotor electrical angle.

number  lxs,              # Stator winding mutual inductance, lxs = -0.5*lms
        lss,              # Stator winding total self inductance, lss = lls + lms
        lrr,              # Rotor winding total self inductance, lrr = llr + lmr
        lmrs,             # Maximum stator to field winding mutual inductance,
                          # lmrs = sqrt(lmr*lms)
        p2,               # Equals p/2, number of pole pairs
        lmrs_p2,          # Used for scaling the torque, lmrs_p2 = lmrs * p2.
        nxtstp,           # Next Step limit, depends on angular velocity and p.
        sprad,            # Sample points for the rotor angle.
        nsrad,            # Newton Steps for the rotor angle.
        theta0_eff,       # Effective (mks equivalent) of theta0.
        wt

<consts.sin

number ang2pi3            # Equal to 2*math_pi/3.
```

```
        phi_b,              # Winding b stator flux linkage.
        phi_c,              # Winding c stator flux linkage.
        phi_fld,                    # Field winding flux linkage.
        phinew

val l   lfa,       # Time varying mutual inductance, field to stator "a" winding.
        lfb,       # Time varying mutual inductance, field to stator "b" winding.
        lfc        # Time varying mutual inductance, field to stator "c" winding.

var i ia           # Phase a stator current.
var i ib           # Phase b stator current.
var i ic           # Phase c stator current.
var i ifld         # Field current.
var i iemf

val v va           # Winding a total voltage, v(asp) - v(sn).
val v vb           # Winding b total voltage, v(bsp) - v(sn).
val v vc           # Winding c total voltage, v(csp) - v(sn).
val v vfld         # Field winding total voltage, v(fp) - v(fm).
val v vemf

val w_radps wr              # Angular velocity of rotor shaft.
val w_radps abswr           # Absolute value of wr.

var ang_rad thetar          # Rotor angular position.
val ang_rad thetae          # Rotor electrical angle, = thetar*p/2.
val tq_Nm te                # Electromagnetic torque on the rotor.

parameters {
        ang2pi3=2*math_pi/3          2/3
        lxs = -0.5*lms*.666666
        lss = lls + lms*.666666
        lrr = llr + lmr
        lmrs = kmult*sqrt(lmr*lms*.66666)/sqrt(.6666666)
        p2 = p/2
        lmrs_p2 = lmrs * p2
        sprad = 200u/p2
        nsrad = 2/p2
        sp = [(-1g,sprad),(-1meg,sprad),(-1k,sprad),(-1,sprad),(0,sprad),
                (1,sprad),(1k,sprad),(1meg,sprad),(1g,0)]
        ns = [(-1g,nsrad),(-1meg,nsrad),(-1k,nsrad),(-1,nsrad),
                (0,nsrad),(1,nsrad),(1k,nsrad),(1meg,nsrad),(1g,0)]
        nxtstp = 0.1/p2
        if (units == mks | units == cgs) {
                theta0_eff = theta0
                }
        else if (units == fps | units == ios) {
                theta0_eff = theta0*0.017453293 # Convert deg to rad.
                }
        }

values {
        # Winding voltages
        va = v(asp) - v(bsp)
        vb = v(bsp) - v(csp)
        vc = v(csp) - v(asp)
```

```
        if (va > 500) va = 500
        if (vb > 500)  vb = 500
        if (vc > 500) vc = 500
        if (va < -500) va = -500
        if (vb < -500)  vb = -500
        if (vc < -500) vc = -500

        # Calculate rotor velocity:
        wr = shaft * 2 * 3.1415926 / 60
#       wr = w_radps(shaft) * 2 * 3.1415926 / 60
#       wr = w_radps(shaft)
#       takes shaft in rpm and creates wr in radians/second

        abswr = abs(wr)

        # Calculate electrical angle
        thetae = thetar*p2

        # Control the simulator step size
        if (abswr > 1) step_size = nxtstp/abswr
        else step_size = nxtstp

        # Calculate time varying mutual inductances
        lfa = lmrs*cos(thetae)
        lfb = lmrs*cos(thetae - ang2pi3)
        lfc = lmrs*cos(thetae + ang2pi3)

        # Calculate stator flux levels
        phi_a   = (lss)*ia + lxs*(ib+ic) + lfa*(ifld) ### + ifld_res)
        phi_b   = (lss)*ib + lxs*(ia+ic) + lfb*(ifld) ### + ifld_res)
        phi_c   = (lss)*ic + lxs*(ia+ib) + lfc*(ifld) ### + ifld_res)
        phi_fld = lfa*ia + lfb*ib + lfc*ic + lrr*(ifld) ### + ifld_res)
        phinew = lfa*ifld


        # Calculate the torque contributed to the rotor.
        ##### Need to add ifld_res to ifld in torque eqn if used.
        te = -lmrs_p2*ifld*(ia*sin(thetae) + ib*sin(thetae-ang2pi3) +
                        ic*sin(thetae+ang2pi3))
        }

control_section{
        sample_points(thetar,sp)
        newton_step(thetar,ns)
        }

equations {
        i(asp->bsp) += ia
        i(bsp->csp) += ib
        i(csp->asp) += ic
        i(fp->fm)  += ifld
        i(emf->com) += iemf
        tq_Nm(shaft) += te

        thetar:  d_by_dt(thetar) = wr*(1-dc_domain) +
                        (thetar-theta0_eff)*dc_domain

        ia: va = rs*ia + d_by_dt(phi_a)
```

```
        ib: vb = rs*ib + d_by_dt(phi_b)
        ic: vc = rs*ic + d_by_dt(phi_c)
        ifld: vfld = rr*ifld + d_by_dt(phi_fld)
        iemf: vemf = d_by_dt(phinew)
        }

moi_w.1          shaft          =j=j, units = units
damper_w.1       shaft    0     =d=d, units = units
}
#########
######### End of internal 3 phase alt
#########

# Netlist of the alternator model
alt3p.1 a b c fp fm shaft emf com =  kmult=kmult,rs=rs,lls=lls,lms=lms,
                                     rr=rr,llr=llr,lmr=lmr,
                                     p=p,d=d,j=j,theta0=theta0,
                                     units=units

}
```

Received June 23-2000 Friday

June 23

Yilmaz,

Here is a detailed response to your fax. I am asking for several minor revisions in your code and want to see some fresh output **today** in time to consider it and respond back to you before the weekend. Call me AFTER you have tried all the suggestions.

First, I explain why Ron and I do not have the errors you suspect.

I then point out a few items for you to try which may fix your code. There is at least one specific error to be fixed which is significant, and there is at least one qualitative error which begs for an explanation. I ask for several specific responses:

1. Your time step and where it is in your code. If it is above 1 microsecond, produce a fresh set of runs at 3000 rpm with a 1 microsecond time step (see item 4 first).
2. A clarification on your diode bridge model (ideal, dropless, lossless????)
3. An explanation of how your model can produce variation for phase voltage in the region where is should be absolutely constant at 44V (assuming 1V diodes). I suspect your time step.
4. A fresh set of runs at 3000 rpm with a 44V bus voltage (assuming I am right that you really have no diode model as such but a perfect dropless bridge in the code right now.)

Response concerning your fax concerning the Energenix results :

I will start with p(5), which was an early version of our code. The items you question involving "+ ifld_res" are not an issue. This was an artifact from a previous author at Analogy. Anything following a "#" sign in MAST is a comment and not active code.

Explanation of the line

ia: va = rs*ia +d_by_dt(phi_a)

In Saber, this is the equation section. Basically, Saber works on the principle of writing the voltage/current equation for each circuit leg and then putting together a netlist and insisting that the current sum at each node of the network is zero (Kirchoff's law).

There is a major difference between your coding approach and the general Saber netlist approach which is causing a misunderstanding for you. In your approach, the diode bridge and the generator (stator circuits) are rolled into one set of equations with multiple cases to consider.

The Energenix Saber generator model you saw is just that, only the generator itself. This does not include the regulator or the rectifier or the bus to which it connects. Thus, you are not seeing the models for the six diodes which connect on one side to the 3-phase

generator delta winding and on the other side to the DC bus (which is 42V for these runs).

Thus, the Energenix method does have the 42V in the equations. You just do not see it in the generator model. This same generator model can be connected to any 3phase input model and perform appropriately.

Response concerning Yilmaz method :

I cannot explain the similarity which you have observed with some numbers, but at this point I think we need to move past that as I do not think it to be significant. First we must deal with at least one obviously quantitative error and at least one serious qualitative error in your code. Then let us re-evaluate after you provide fresh output.

Error #1: You are using a bus voltage of 42V and 0V for the diodes. Please note I have always been saying that you should use a 1 V drop for each diode. Thus, to match results, you need to use a 44V bus voltage if your code does not model diodes. This will change your phase currents but not the EMF.

Error #2: This is a further comment on the traces you sent to me earlier in the week. When the phase voltage hits its peak (should be 44V), there should be NO variation whatsover whereas you produce a somewhat jagged response. This indicates the possibility that you are using too large time step (What is your time step – I cannot find it in your code???). My conjecture may be wrong, but it is not debatable that this plot should be absolutely flat at 44V with no variation whatsoever. (My solver is automated to vary the time step as the equations permit, but the baseline time step I am specifying is 1 microsecond; at times, the solver uses an even smaller time step.)

Comment: I have revisited your code. Please tell me if this is correct. You are presently not using a diode model, but rather in the section with IADCGEN you are effectively using a "dropless" diode bridge? If this is the case, you definitely need to have a 44V bus voltage. Moreover, if this is the case, my earlier hypothesis concerning your diode model becomes moot.

June 29

Yilmaz,

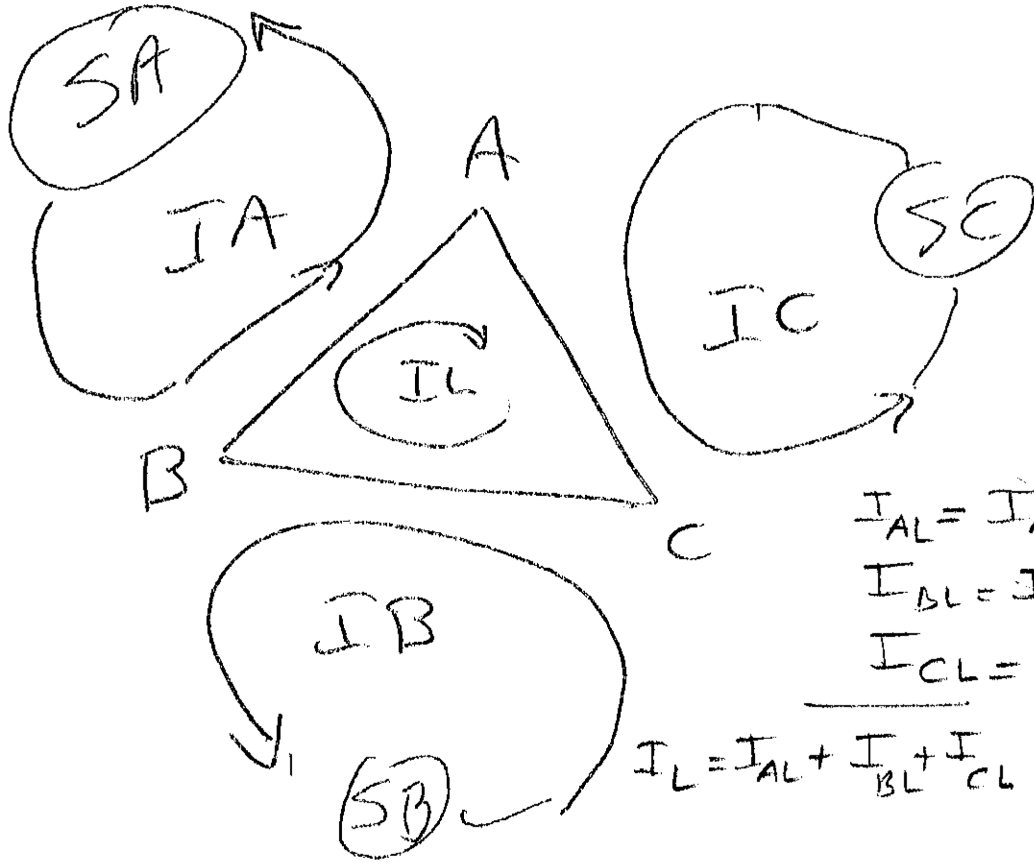
I suspect your methodology cannot easily be fixed. The problem stems from the fact that you need to patch together currents at nodes and worry about conservation of current at each node. This is not easily done with logic based upon voltages. The result is that you are going down the path of cases which should not occur.

Let me offer a complete theory and Fortran code which does work. This is identical to what should be occurring in Saber, but is all accessible for you without dealing with other Saber models and netlisting. All you have to do is implement this successfully in CSSL and you will have a functional generator model. You will have to deal yourself with the case when you are below cutin and (potentially a little different), just above cutin.

If you check any circuits book, they always take the approach of defining loop currents. Then all the equations are solved for loop currents and then the voltages are computed therefrom. The reason this approach is always taken is that then you always have continuity of current at nodes. This is the basis of my theory.

The theory is laid out in the attached pages. Also, the complete Fortran code is provided.

John



$I_{AL} = I_A - I_C$

$I_{BL} = I_B - I_A$

$I_{CL} = I_C - I_B$

$I_L = I_{AL} + I_{BL} + I_{CL} = 0.0$

4 Loop Currents

Note that IL is superfluous. You can make IL anything and compensate by altering IA, IC, & IB the other way. This is because the net current out at A is IA-IC, so the alteration cancels out. Thus, set ILoop = 0 & proceed.

```
      program test1

      real*8 omega, ls, lm, gencoef, ifgen
      real*8 t, deltat, rpm, pi
      real*8 vagen, vbgen, vcgen
      real*8 iadot, ibdot, icdot
      real*8 va, vb, vc
      real*8 ia, ib, ic
      real*8 vd, vbat, res, isgen, ilinea

c     initialize variables
      deltat = 1.0E-5
      t = -deltat
      rpm = 3000
      pi = ACOS(-1)
c      print*,"pi = ", pi
      lm =  .0162
      ifgen = 4.9
c     the 6 is the stator multiplier (12 poles) times generator rpm
      omega =   2*pi*rpm*6/60
      gencoef = -omega * ifgen * lm
      vd = 1.0
      vbat = 42
      ls = .002275
c     stator resistance in ohms
      res = .29

      vagen = 0.
      vbgen = 0.
      vcgen = 0.
      iadot = 0.0
      ibdot = 0.0
      icdot = 0.0
      ia = 0.0
      ib = 0.0
      ic = 0.0

c     Main time loop starts here
      do 2 k = 1, 2000
      do 1 j = 1, 10
      t = t + deltat

      ia = ia + deltat * iadot
      ib = ib + deltat * ibdot
      ic = ic + deltat * icdot

      vagen = gencoef * sin(omega * t)
      vbgen = gencoef * sin(omega * t - (2 * pi /3))
      vcgen = gencoef * sin(omega * t + (2 * pi /3))

c     This section assigns the rectifer voltage for that particular loopl
c     Note that all possibilities are available.
      if (ia .gt. ic) va = vbat + vd
c     really, if this case occurs, should be pro-rated and not 21
      if (ia .eq. ic) va = vbat/2
      if (ia .lt. ic) va = -vd
      if (ib .gt. ia) vb = vbat + vd
```

```
      if (ib .eq. ia) vb = vbat/2
      if (ib .lt. ia) vb = -vd
      if (ic .gt. ib) vc = vbat + vd
      if (ic .eq. ib) vc = vbat/2
      if (ic .lt. ib) vc = -vd

      iadot = (1/ls)*(vagen - va + vb - ia*res)
      ibdot = (1/ls)*(vbgen - vb + vc - ib*res)
      icdot = (1/ls)*(vcgen - vc + va - ic*res)

      isgen = 0
      if (ia .gt. ic) isgen = isgen + ia - ic          Iae
      if (ib .gt. ia) isgen = isgen + ib - ia          Ibe
      if (ic .gt. ib) isgen = isgen + ic - ib          Ice
1     continue
      ilinea = ia - ic
      print*, t, ia, ilinea
2     continue

      stop
      end
```